# LNKnet: Neural Network, Machine-Learning, and Statistical Software for Pattern Classification

Richard P. Lippmann, Linda Kukolich, and Elliot Singer

■ Pattern-classification and clustering algorithms are key components of modern information processing systems used to perform tasks such as speech and image recognition, printed-character recognition, medical diagnosis, fault detection, process control, and financial decision making. To simplify the task of applying these types of algorithms in new application areas, we have developed LNKnet—a software package that provides access to more than 20 pattern-classification, clustering, and feature-selection algorithms. Included are the most important algorithms from the fields of neural networks, statistics, machine learning, and artificial intelligence. The algorithms can be trained and tested on separate data or tested with automatic cross-validation. LNKnet runs under the UNIX operating system and access to the different algorithms is provided through a graphical point-and-click user interface. Graphical outputs include two-dimensional (2-D) scatter and decision-region plots and 1-D plots of data histograms, classifier outputs, and error rates during training. Parameters of trained classifiers are stored in files from which the parameters can be translated into source-code subroutines (written in the C programming language) that can then be embedded in a user application program. Lincoln Laboratory and other research laboratories have used LNKnet successfully for many diverse applications.
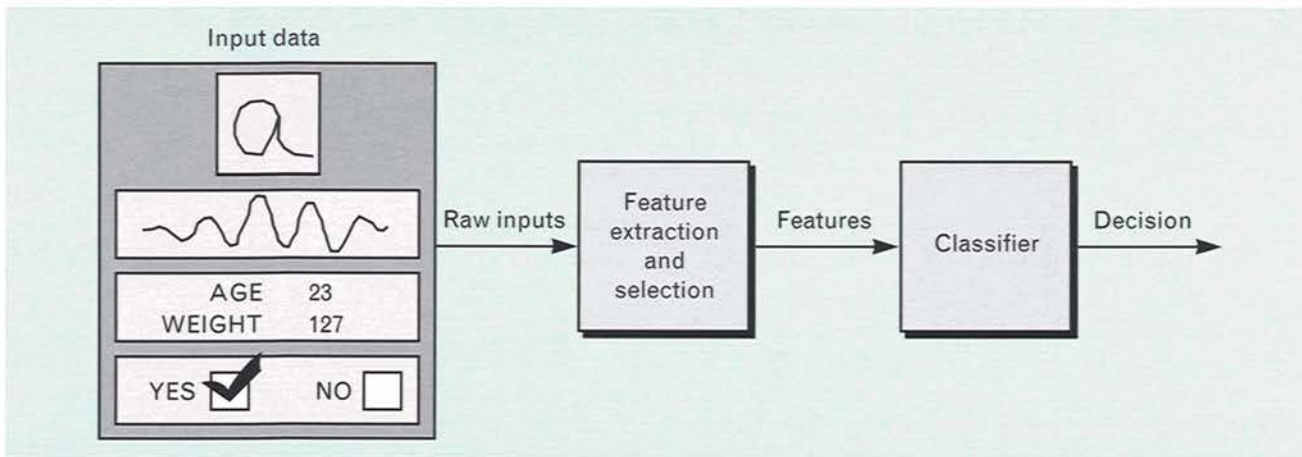
PATTERN-CLASSIFICATION ALGORITHMS are difficult to implement in a manner that simplifies the task of training, evaluating, and applying them correctly to new problems. At Lincoln Laboratory and other sites, researchers were spending an excessive amount of programming time to implement and debug the same classification algorithms and to create complex command scripts to run experiments. Classifiers were often implemented by different programmers using idiosyncratic programming conventions, user interfaces, and data interfaces. This lack of standardization made it difficult to compare classifiers and to embed them in user application programs. Consequently, to prevent this duplicate programming and to simplify the task of applying classification algorithms, we developed LNKnet—a software package that provides access to more than 20 pattern-classification, clustering, and feature-selection algorithms. Included are the most important algorithms from the fields of neural networks, statistics, machine learning, and artificial intelligence. Access to the different algorithms is provided through a point-and-click user interface, and graphical outputs include two-dimensional (2-D) scatter and decision-region plots and 1-D plots of data histograms, classifier outputs, and error rates during training. (Note: The acronym LNK stands for the initials of the last names of the software's three principal programmers—Richard Lippmann, Dave Nation, and Linda Kukolich).

This article first presents an introduction to pat-

| | | Form Approved<br>*OMB No. 0704-0188* |
|---|---|---|
| **Report Documentation Page** | | |

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

| 1. REPORT DATE<br>**1993** | 2. REPORT TYPE | 3. DATES COVERED<br>**00-00-1993 to 00-00-1993** |
|---|---|---|
| 4. TITLE AND SUBTITLE<br>**LNKnet: Neural Network, Machine-Learning, and Statistical Software for Pattern Classification** | | 5a. CONTRACT NUMBER |
| | | 5b. GRANT NUMBER |
| | | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | | 5d. PROJECT NUMBER |
| | | 5e. TASK NUMBER |
| | | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**Massachusetts Institute of Technology,Lincoln Laboratory,244 Wood Street,Lexington,MA,02420** | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT<br>**Approved for public release; distribution unlimited** | | |
| 13. SUPPLEMENTARY NOTES | | |
| 14. ABSTRACT | | |
| 15. SUBJECT TERMS | | |

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **Same as Report (SAR)** | **20** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

**FIGURE 1.** A simple pattern-classification system with image, waveform, categorical, and binary inputs.

tern classification and then describes the LNKnet software package. The description includes a simple pattern-classification experiment that demonstrates how LNKnet is applied to new databases. Next, this article describes three LNKnet applications. In the first application, LNKnet radial-basis-function subroutines are used in a hybrid neural-network/hidden-Markov-model isolated-word recognizer. The second application is an approach to secondary testing for wordspotting in which LNKnet multilayer perceptron classifiers are accessed through the system's point-and-click interface. In the final application, LNKnet is used to develop a system that learns in real time the strategy a human uses to play an on-line computer game. This strategy-learning system was developed with the LNKnet point-and-click interface and then implemented for real-time performance with the LNKnet multilayer perceptron subroutines.

## Introduction to Pattern Classification

The purpose of a pattern classifier is to assign every input pattern to one of a small number of discrete classes, or groups. For example, if the input to a classifier is the enlarged image of cells from a Pap smear, the output classes could label the cells as normal or cancerous. Figure 1 shows a block diagram of a simple pattern-classification system. Inputs from sensors or processed information from computer databases are fed into a preprocessor that extracts measurements or *features*. The features simplify the classification task: irrelevant information is eliminated

by focusing only on those properties of the raw inputs which are distinguishable between classes. The input feature measurements $x_1$, $x_2$, $x_3$, . . . , $x_D$ form a feature vector $\mathbf{X}$ with $D$ elements in each vector. The feature vectors, or patterns, are fed into a classifier that assigns each vector to one of $M$ prespecified classes denoted $C_i$. Given a feature vector, a typical classifier creates one *discriminant function,* or output $y_i$, per class. The decision rule that most classifiers use is to assign the feature vector to the class corresponding to the discriminant function, or output, with the highest value. All classifiers separate the space spanned by the input variables into *decision regions,* which correspond to regions where the classification decision remains constant as the input features change.

The three major approaches to developing pattern classifiers are the *probability-density-function (PDF), posterior-probability,* and *boundary-forming* strategies. These approaches differ in the statistical quantity that their outputs model and in the procedures they use for classifier training: PDF classifiers estimate class likelihoods or probability density functions, posterior-probability classifiers estimate Bayesian *a posteriori* probabilities [1] (hereafter referred to as posterior probabilities), and boundary-forming classifiers form decision regions. Figure 2 illustrates the shape of these functions for a simple problem with one input feature, two classes denoted A and B, and Gaussian class distributions. The PDF functions formed by statistical classifiers are Gaussian shaped, as shown in Figure 2(a). These functions represent the distribu-

tions of the input feature for the two classes. Posterior probabilities formed by many neural network classifiers have sigmoidal shapes, as shown in Figure 2(b). These functions vary from 0 to 1, their sum equals 1, and they represent the probability of each class, given a specific input value. Finally, the binary indicator outputs of boundary-forming classifiers separate the input into two regions, one for class A and the other for class B, as shown in Figure 2(c).

## A Taxonomy of Pattern Classifiers

Table 1 contains a taxonomy of the most common PDF, posterior-probability, and boundary-forming classifiers. The first three types of classifiers in this table produce continuous probabilistic outputs, while the last two produce binary indicator outputs.

The first row in Table 1 represents conventional PDF classifiers [2, 3], which model distributions of pattern classes separately through the use of parametric functions. In the decision-region diagram, the green and blue dots represent the means of classes A and B, respectively, the circles denote the respective standard deviations for the two classes, and the black line represents the boundary between decision regions for the two classes.
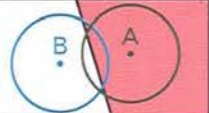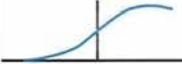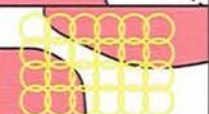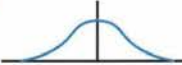
The next two rows in Table 1 contain two types of neural network posterior-probability classifiers. *Global* neural network classifiers [4–6]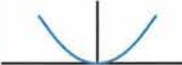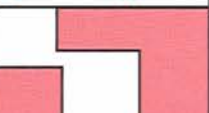 form output discriminant functions from internal computing elements or nodes that use sigmoid or polynomial functions having high nonzero outputs over a large region of

the input space. In the decision-region diagram, the three black lines represent half-plane decision-region boundaries formed by sigmoid nodes. Global neural network classifiers include multilayer perceptrons (MLP) trained with back propagation, Boltzmann machines, and high-order polynomial networks. *Local* neural network classifiers [7] form output discriminant functions from internal computing elements that use Gaussian or other radially symmetric functions having high nonzero outputs over only a localized region of the input space. In the decision-region diagram, the yellow cells represent individual computing elements and the two black curves represent decision-region boundaries. Local neural network classifiers include radial basis function (RBF) and kernel discriminant classifiers. These two types of classifiers make no strong assumptions concerning underlying distributions, they both form complex decision regions with only one or two hidden layers, and they both are typically trained to minimize the mean squared error between the desired and actual network outputs.

The bottom two rows of Table 1 contain boundary-forming classifiers. *Nearest neighbor* classifiers [2, 7] perform classification based on the distance between a new unknown input and previously stored exemplars. In the decision-region diagram, the blue crosses and green diamonds represent training patterns from two different classes, and the two black jagged lines represent the boundaries between those two classes. Nearest neighbor classifiers, which in-
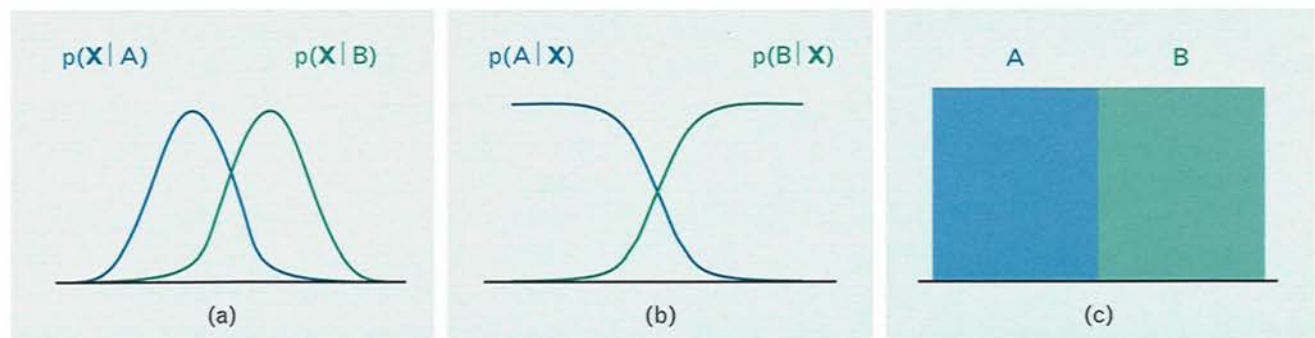


**FIGURE 2.** Discriminant functions formed by (a) probability-density-function (PDF), (b) posterior-probability, and (c) boundary-forming classifiers for a problem with one input feature and two classes A and B. Note that PDF classifiers estimate likelihoods, posterior-probability classifiers estimate posterior probabilities, and boundary-forming classifiers create decision regions.

| Table 1. A Pattern-Classification Taxonomy | | | |
|---|---|---|---|
| Type of Classifier | Decision Region (shaded in red) | Computing Element | Representative Classifiers |
| PDF | | Distribution dependent | Gaussian, Gaussian mixture |
| Global | | Sigmoid | Multilayer perceptron, high-order polynomial network |
| Local | | Kernel | Radial basis function, kernel discriminant |
| Nearest Neighbor | | Euclidean norm | K-nearest neighbor, learning vector quantizer |
| Rule Forming | | Threshold logic | Binary decision tree, hypersphere |

Note: For a description of the five different types of classifiers listed, see the main text.

clude conventional *K*-nearest neighbor (KNN) classifiers and neural network learning vector quantizer (LVQ) classifiers, train extremely rapidly but they can require considerable computation time on a serial processor as well as large amounts of memory. *Rule-forming* classifiers [2, 7–11] use threshold-logic nodes or rules to partition the input space into labeled regions. An input can then be classified by the label of the region where the input is located. In the decision-region diagram for rule-forming classifiers in Table 1, the black lines represent the decision-region boundaries formed by threshold-logic nodes or rules. Rule-forming classifiers have binary outputs and include binary decision trees, the hypersphere classifier, perceptrons with hard-limiting nonlinearities trained with the perceptron convergence procedure, sigmoidal or RBF networks trained with differential training, and many machine-learning approaches that result in a small set of classification rules.

No one type of classifier is suitable for all applications. PDF classifiers provide good performance when the probability density functions of the input features

are known and when the training data are sufficient to estimate the parameters of these density functions. The most common PDF classifier is the Gaussian classifier. The use of Gaussian density functions with common class covariance matrices is called Linear Discriminant Analysis (LDA) because the discriminant functions reduce to linear functions of the input features. LDA provides good performance in many simple problems in which the input features do have Gaussian distributions. But, when the training data are limited or when the real-world feature distributions are not accurately modeled by Gaussian distributions, other approaches to classification provide better performance.

Global and local neural network classifiers are both suitable for applications in which probabilistic outputs are desired. Global neural network classifiers that use sigmoid nodes are most suitable for applications such as speech recognition and handwritten-character recognition in which a large amount of training data is available, and in which the training time can be slow but the speed of recognition during use must be

fast. These classifiers are also well suited for implementation in parallel VLSI hardware that supports the simple types of computation required by multilayer sigmoid networks. Local neural networks such as RBF classifiers are most suitable when the input features have similar scales and do not differ qualitatively and when shorter training times are desired at the expense of slightly longer classification times.

Nearest neighbor classifiers are best suited for problems in which fast training and adaptation are essential but in which there is sufficient memory and enough computational power to provide classification times that are not too slow.

Finally, rule-based classifiers and decision trees are most suitable when a minimal-sized classifier is desired that can run extremely fast on a uniprocessor computer and when simple explanations for classifier decisions are desired.

## Overview of LNKnet

LNKnet was developed to simplify the application of the most important neural network, statistical, and machine learning classifiers. We designed the software so that it could be used at any one of the three levels shown in Figure 3.
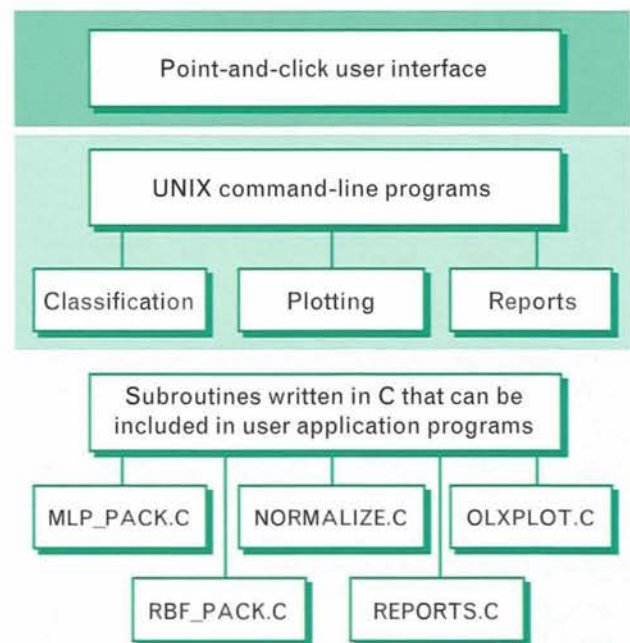
The point-and-click graphical user interface can be used to experiment rapidly and interactively with classifiers on new databases. This approach is the simplest way to apply classification algorithms to new databases. After converting a database into a simple ASCII format, a user can run experiments by making the appropriate selections in LNKnet windows with a mouse and keyboard. A complex series of experiments on a new moderate-sized database (containing thousands of patterns) can be completed in less than an hour. Use of the point-and-click interface does not require any knowledge of UNIX shell scripts, C programming, or the way in which LNKnet algorithms are implemented.

Users who want to execute long batch jobs can edit and run the shell scripts produced by the point-and-click interface. This approach, which requires an understanding of shell scripts and the arguments to LNKnet programs, simplifies the repetitive application of the same algorithm to many data files and automates the application of LNKnet when batch-

mode processing is desired.

Finally, users with knowledge of C programming can work at the source-code level. At this level, C source code that implements LNKnet subroutines and libraries can be embedded in a user application program. We have simplified this procedure with filter programs. The programs read in LNKnet parameter files defining trained classifiers and create C source-code subroutines to implement those classifiers. These C source-code subroutines can be embedded in a user application program.

LNKnet contains more than 20 neural network, pattern-classification, and feature-selection algorithms (Table 2), each of which can be trained and then tested on separate data or tested with automatic cross-validation. The algorithms include classifiers that are trained with labeled data under supervision, classifiers that use clustering to initialize internal parameters and then are trained with supervision, and clustering algorithms that are trained with unlabeled data without supervision. Algorithms for Canonical Linear Dis-



**FIGURE 3.** The three levels of using the LNKnet software package. Researchers can access LNKnet either through the point-and-click user interface, or by manually editing shell scripts containing LNKnet commands to run batch jobs, or by embedding LNKnet subroutines in application programs.

**Table 2. LNKnet Algorithms**

| | Supervised Training | Combined Unsupervised-Supervised Training | Unsupervised Training (Clustering) |
|---|---|---|---|
| Neural Network Algorithms | Multilayer perceptron (MLP) <br> Adaptive step-size MLP <br> Cross-entropy MLP <br> Differential trained MLP <br> Hypersphere classifier | Radial basis function (RBF) <br> Incremental RBF (IRBF) <br> Differential IRBF <br> Learning vector quantizer (LVQ) <br> Nearest-cluster classifier | Leader clustering |
| Conventional Pattern-Classification Algorithms | Gaussian linear discriminant <br> Quadratic Gaussian <br> *K*-nearest neighbor (KNN) <br> Condensed KNN <br> Binary decision tree | Gaussian-mixture classifier <br> Diagonal/full covariance <br> Tied/per-class centers | *K*-means clustering <br> Estimate-Maximize (EM) clustering |
| Feature-Selection Algorithms | Canonical Linear Discriminant Analysis (LDA) <br> KNN forward and backward search | | Principal Components Analysis (PCA) |

criminant Analysis and Principal Components Analysis have been provided to reduce the number of input features through the use of new features that are linear combinations of old features. KNN forward and backward searches have been included to select a small number of features from among the existing features. Descriptions and comparisons of these algorithms are available in References 2, 6, 9, and 12 through 21.

All LNKnet software is written in C and runs under the UNIX operating system. The graphical user interface runs under MIT X or Sun Microsystem's OpenWindows. (Note: Reference 14 includes a comprehensive description of this user interface.) Graphical outputs include 2-D scatter and decision-region plots and overlaid *internals* plots that illustrate how decision regions were formed. Also available are 1-D histogram plots, 1-D plots of classifier outputs, and plots showing how the error rate and cost function change during training. Standard printouts include *confusion matrices,* summary statistics of the errors for each class, and estimates of the binomial standard

deviations of error rates.

LNKnet allows the training and testing of large classifiers with numerous input features and training patterns. Indeed, we have trained and tested classifiers having up to 10,000 parameters, or weights, and we have trained classifiers with more than 1000 input features and more than 100,000 training patterns. During training and testing, all control screens are saved automatically so that they can be restored at a later time if desired. This feature allows the continuation and replication of complex experiments. Parameters of trained classifiers are stored in files and can be used by code-generation filters to generate freestanding classifier subroutines that can then be embedded in user code.

## Components of Pattern-Classification Experiments

The LNKnet graphical interface is designed to simplify classification experiments. Figure 4 shows the sequence of operations involved in the most common

classification experiment. At the beginning of each experiment, a classification algorithm is selected, and parameters that affect the structure or complexity of the resulting classifier are also chosen. These parameters, which are sometimes called *regularization parameters,* include the number of nodes and layers for MLP classifiers and trees, the training time and value of weight decay for MLP classifiers, the number of mixture components for Gaussian-mixture classifiers, the type of covariance matrix used (full or diagonal, grand average across or within classes) for Gaussian or Gaussian-mixture classifiers, the value of $K$ for KNN classifiers, the number of centers for RBF classifiers, and the number of principal component features used as inputs to a classifier.

A database for a classification experiment typically contains three separate sets of data: training data, evaluation data, and test data. As shown in Figure 4, training data are used initially to train the internal weights or trainable parameters in a classifier. The error rate of the trained classifier is then evaluated with the evaluation data. This procedure is necessary because it is frequently possible to design a classifier that provides a low error rate on training data but that does not perform as well on other data sampled from the same source. Repeated evaluations are followed by retraining with different values for regularization parameters. The regularization parameters adjust the complexity of the classifier, making the classifier only as complex as necessary to obtain good classification performance on unseen data. After all regularization
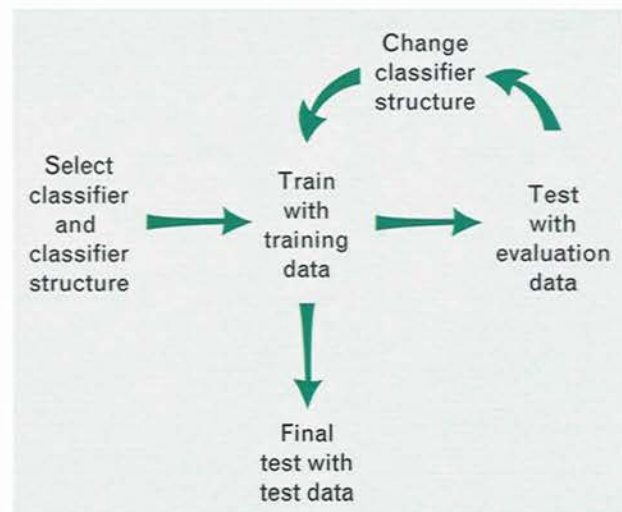


**FIGURE 4.** Components of a classification experiment.

parameters have been adjusted, the classifier generalization error rate on unseen data is estimated with the test data.

One of the most important features of LNKnet is that it includes the ability to normalize input data and to select a subset of input features for classification (Figure 5). Normalization algorithms available in LNKnet include simple normalization (each feature is normalized separately to zero mean, unit variance), Principal Components Analysis (PCA), and Linear Discriminant Analysis (LDA) [2, 22]. Feature-selection algorithms include forward and backward searches [22], which select features one at a time based on the increase or decrease in the error rate measured with a
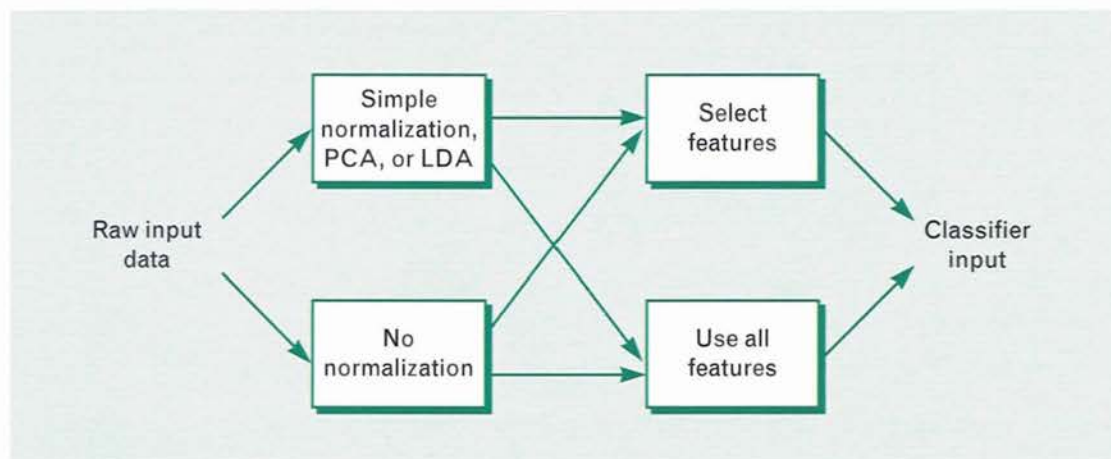


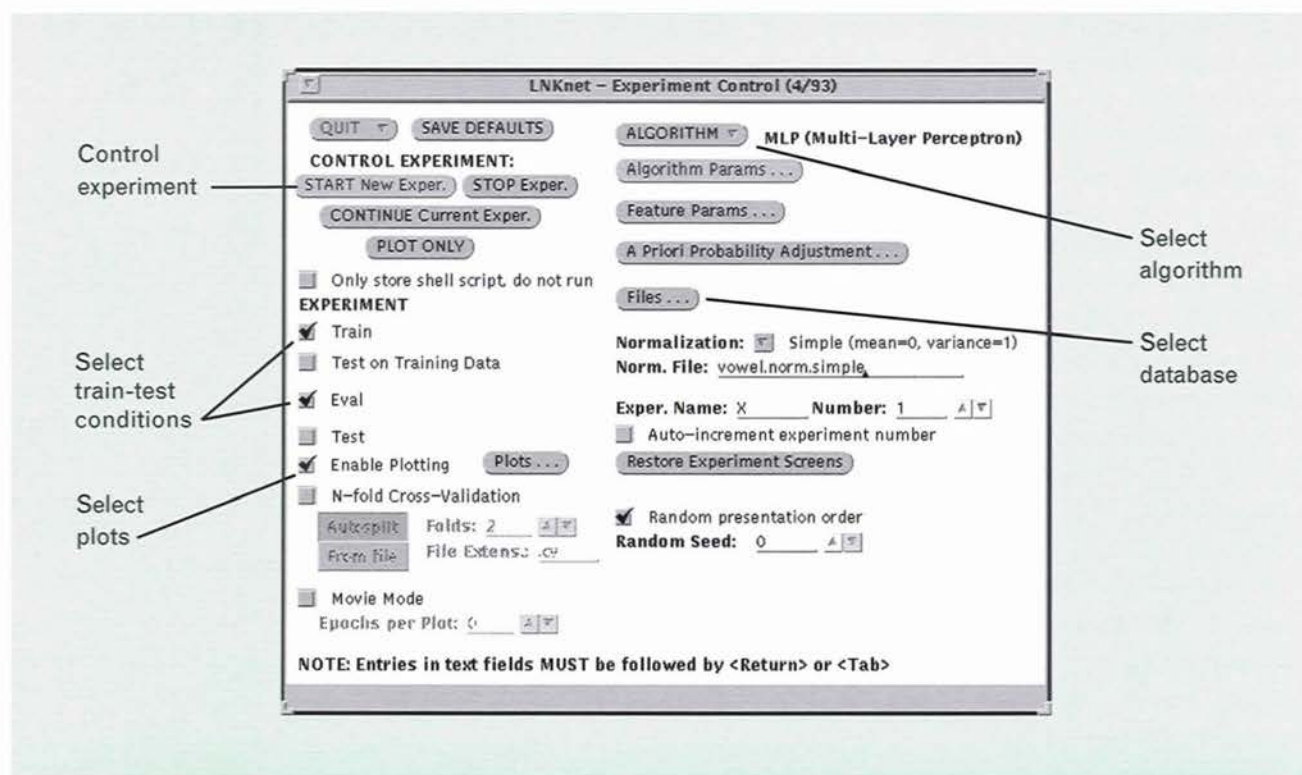**FIGURE 5.** Feature selection and normalization available in LNKnet.

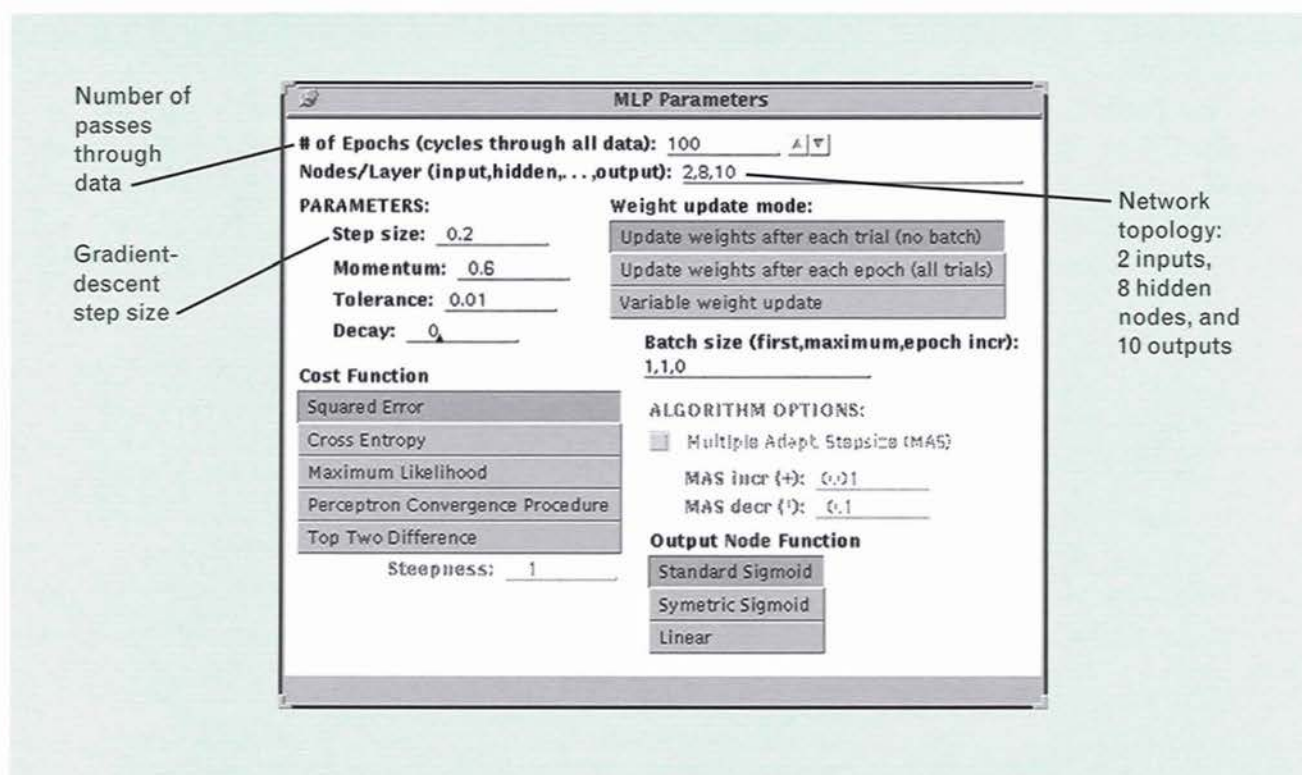**FIGURE 6.** Main LNKnet window used in the vowel-classification experiment.



**FIGURE 7.** "MLP Parameters" window used in the vowel-classification experiment.

nearest neighbor classifier and leave-one-out cross-validation. A forward or backward search, a PCA, or an LDA can be used to obtain a list of features ordered in terms of their presumed importance. From this list, a subset of features can be selected for use in the classification process. This subset can be the first (and presumably most important) features or a selection of unordered features. A user can skip either the normalization or feature-selection steps, thus allowing the classifier to use any or all features of the raw data or the normalized data, as shown in Figure 5.

## A Vowel-Classification Experiment

The use of LNKnet to run experiments is best illustrated by an example. The experiment presented here uses vowel data from a study performed by G.E. Peterson and H.L. Barney [23], in which the first and second vocal-tract resonance frequencies were measured from the spectra of 10 vowels produced by 76 men, women, and children saying the following words: head, hid, hod, had, hawed, heard, heed, hud, who'd, and hood. These two formant frequencies $x_1$ and $x_2$, which are known to be important for identifying vowel sounds, were used as inputs to a classifier with 10 classes consisting of the 10 vowels. Selecting parameters on LNKnet windows and running the vowel-classification experiments described in the following paragraphs took less than 3 min on a Sun Sparc 10 workstation.

Figure 6 shows the main LNKnet control window that was used in our vowel-classification experiment. To set up the experiment, we selected the vowel database, chose the MLP classifier and its structure, checked the "Train," "Eval," and "Enable Plotting" boxes in the main window, and selected the desired types of plots. The database, the algorithm parameters, and the types of plots were selected with other windows that appeared when the appropriate buttons were selected in the main window. For example, the "Algorithm Params . . ." button in the upper right of the main window brought up the "MLP Parameters" window shown in Figure 7. The "MLP Parameters" window was used to select the network structure (2 inputs, 8 hidden nodes, and 10 outputs), the number of times to pass through the entire training dataset during training (100 passes, or epochs), the gradient-descent step size used during training (0.2), the cost function, and other parameters that control the training of MLP classifiers. (Note: For a description of MLP classifiers, see Reference 6.)

LNKnet sets all of these parameters (as well as the parameters in all of the other windows) automatically to the most typical default values so that a user does not have to set each of the parameters manually. A user also has the capability to create new default parameter settings by making the desired selections in all windows, followed by selecting the "SAVE DEFAULTS" button in the upper left of the LNKnet main window. Whenever an experiment is started, the parameter settings for all LNKnet windows are automatically stored in a file so that a user can read in the parameter settings at a later time (e.g., to continue a prior experiment after performing other experiments) by selecting the "Restore Experiment Screens" button in the main window.

Once all classifier parameters have been chosen, a user begins an experiment by selecting the "START" button in the main window. This step first creates a UNIX shell script to run an experiment and then runs the shell script in the background. The results of the experiment are written in a file and printed to the OpenWindows window used to start LNKnet. After each pass through the training data, LNKnet prints the current classification error rate and the current mean squared error.

When training is completed, a summary of the training errors is printed, followed by the confusion matrix and error summary for the evaluation data, as shown in Tables 3 and 4. The confusion matrix contains totals for the number of times the input pattern was from class $C_i$, $1 \leq i \leq M$, and the number of times the decision, or computed class, was from class $C_j$, $1 \leq j \leq M$, over all patterns in the evaluation dataset. (Note: For the ideal case in which LNKnet classifies every input correctly, all of the off-diagonal entries in the confusion matrix would be zero.) Summary statistics contain the number of input patterns in each class, the number of errors and the percent errors for each class, the estimated binomial standard deviation of the error estimate, the root-mean-square (rms) difference between the desired and the actual network outputs for patterns in each class, and the label for

**Table 3. Classification Confusion Matrix for the Vowel-Classification Experiment**

| Desired Class | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 16 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 17 |
| 2 | 0 | 16 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 18 |
| 3 | 0 | 0 | 17 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 20 |
| 4 | 1 | 0 | 0 | 11 | 0 | 1 | 0 | 5 | 0 | 0 | 18 |
| 5 | 0 | 0 | 2 | 0 | 12 | 0 | 0 | 0 | 0 | 2 | 16 |
| 6 | 1 | 1 | 0 | 0 | 0 | 5 | 0 | 2 | 0 | 2 | 11 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 18 | 0 | 0 | 0 | 18 |
| 8 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 17 | 0 | 0 | 18 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 3 | 16 |
| 10 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 2 | 2 | 7 | 14 |
| Total | 18 | 18 | 19 | 11 | 13 | 9 | 20 | 29 | 15 | 14 | 166 |

The column group header reads "Computed Class" spanning columns 1–10.

**Table 4. Error Report for the Vowel-Classification Experiment**

| Class | Number of Patterns | Number of Errors | Percent Errors | Binomial Standard Deviation | rms Errors | Label |
|---|---|---|---|---|---|---|
| 1 | 17 | 1 | 5.88 | ± 5.7 | 0.152 | head |
| 2 | 18 | 2 | 11.11 | ± 7.4 | 0.158 | hid |
| 3 | 20 | 3 | 15.00 | ± 8.0 | 0.159 | hod |
| 4 | 18 | 7 | 38.89 | ± 11.5 | 0.219 | had |
| 5 | 16 | 4 | 25.00 | ± 10.8 | 0.176 | hawed |
| 6 | 11 | 6 | 54.55 | ± 15.0 | 0.263 | heard |
| 7 | 18 | 0 | 0.00 | 0.0 | 0.064 | heed |
| 8 | 18 | 1 | 5.56 | ± 5.4 | 0.122 | hud |
| 9 | 16 | 3 | 18.75 | ± 9.8 | 0.150 | who'd |
| 10 | 14 | 7 | 50.00 | ± 13.4 | 0.259 | hood |
| Overall | 166 | 34 | 20.48 | ± 3.1 | 0.175 | |

each class. In our vowel-classification experiment, the labels were the 10 words used to produce the 10 vowels.

The results of our vowel-classification experiment are shown in Table 4. Note that the overall error rate on the evaluation data was 20.48%, there were approximately equal numbers of patterns for each class, and the classes that caused the most confusions were "heard," "hood," and "had." These results were near the best that can be obtained with this database. The error rate was high (roughly 20%) because we used only two input features, thus ignoring the dynamics of speech production. We also did not consider the gender and age of the talkers.

By checking the appropriate boxes in the LNKnet "Plotting Controls" window, we specified the drawing of three plots. Figure 8 shows the resulting three overlaid 2-D plots: a decision-region plot (the solid colored regions), a scatter plot of the evaluation data (the small white-rimmed squares), and an internals plot (the black lines). The decision-region plot indicates the classification decision formed by the MLP classifier for any input feature vector in the plot area. For example, input feature vectors in the upper right yellow region are classified as the vowel in "had." It should be noted that the values of these features were normalized with simple normalization across all classes. The scatter plot shows the evaluation data, color coded to show the different classes. Thus classification errors are indicated by squares whose colors do not match the background color of the decision-region plot. The internals plot shows how internal computing elements in each classifier form decision-region boundaries. For the MLP classifier, LNKnet draws lines representing hyperplanes defined by nodes in the first hidden layer [6]. (With Gaussian, Gaussian-mixture, and RBF classifiers, LNKnet draws ovals showing the centers and variances of the Gaussian functions used in the classifiers.) These hyperplane lines for the MLP classifier demonstrate how decision-region borders are formed and often help determine the minimum number of hidden nodes that are required. For experiments involving more than two input features, we can create 2-D plots by selecting any two input features of interest and setting the other inputs to fixed values.
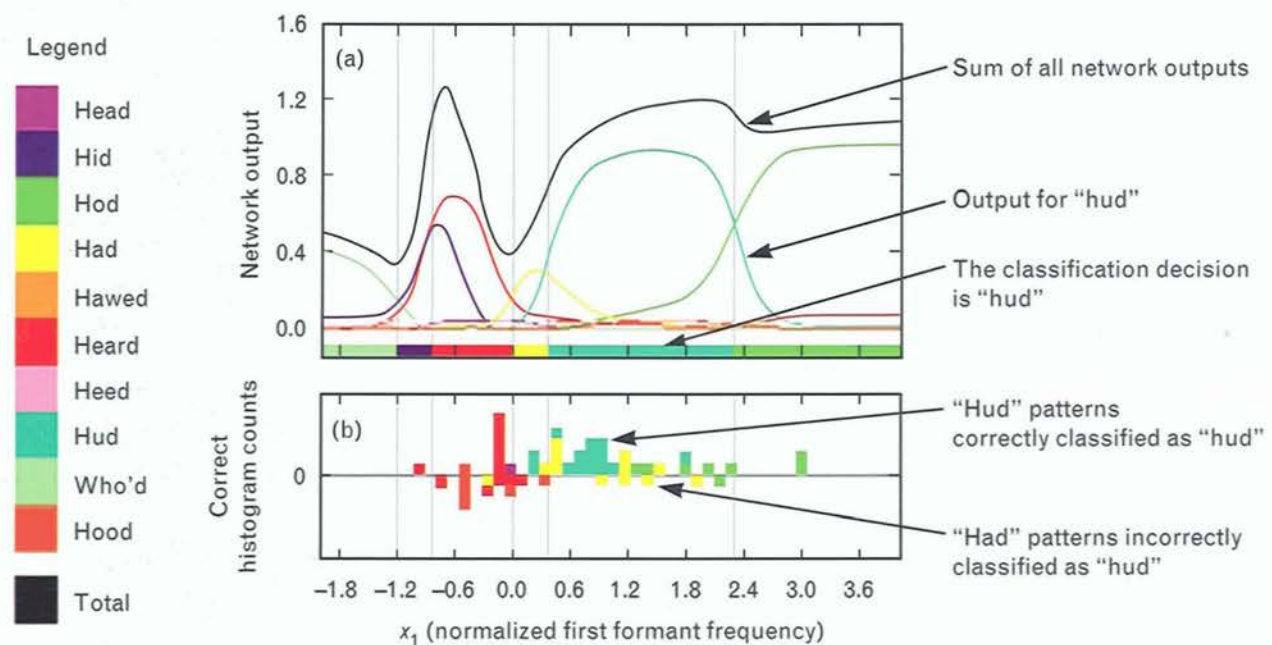
During the vowel-classification experiment, LNKnet also produced profile and histogram plots. Figure 9(a) is a profile of the 10 classifier outputs shown with different colors for the case in which the second input feature $x_2$ is set to 0.0 and the first feature $x_1$ is swept from −2.0 to 4.0. This case corresponds to a plot of the network outputs over a horizontal line ($x_2$ = 0.0) that bisects Figure 8. In Figure 9(a) the sum of all of the 10 outputs is shown in black. This sum will be close to 1.0 for a well-trained classifier that estimates Bayesian posterior class probabilities accurately. A 1-D decision-region plot is provided at the bottom of Figure 9(a) to indicate which class is chosen as the first input feature $x_1$ is swept over the plotted range. Gray vertical lines, drawn wherever there is a change in the choice of class, indicate the decision-region boundaries. Figure 9(b) is a histogram in which the colored squares above the horizontal axis represent patterns that the current model has classified correctly. The squares below indicate misclassified patterns. The squares are color coded by class and only those patterns in the evaluation dataset which are within a prespecified distance of the $x_2$ = 0.0 line in Figure 8 are included in this histogram. Figures 9(a) and 9(b) show the shapes of the discriminant functions formed by the classifier outputs; the plots help users to infer the input ranges over which these functions may be used reliably to estimate posterior probabilities and likelihoods.

Figure 10, the final plot produced during the vowel-classification experiment, shows how the rms error between the desired and actual network outputs decreases during training. In the experiment, 338 unique patterns were presented to LNKnet in random order during each training pass. There were 100 passes through the training data; thus a total of 33,800 training trials were performed, and the rms error was plotted once for each pass. (Note: This training took less than 30 sec on a Sun Sparc 10 workstation.) As can be seen in Figure 10, the rms error dropped from above 0.3 to below 0.2 with most of the reduction occurring early in training. Plots such as Figure 10 are useful to determine whether gradient descent training has converged and to study how changes in step size and other training parameters affect the error convergence.
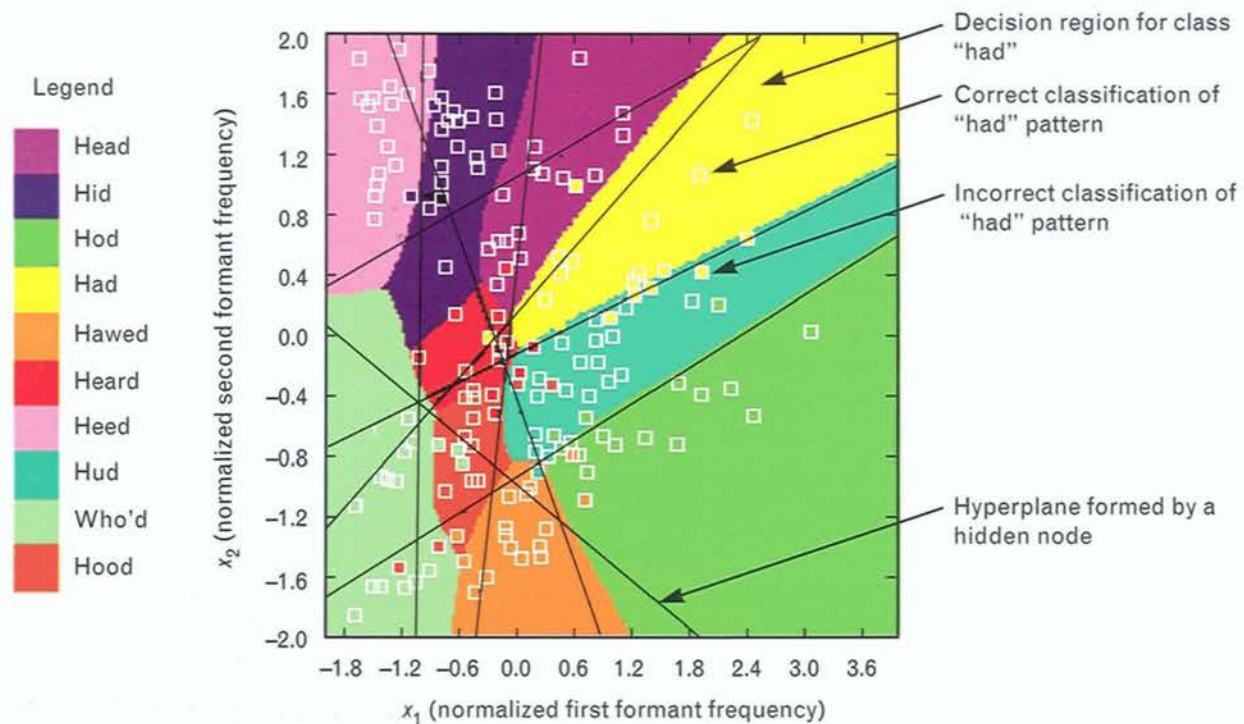
**FIGURE 8.** Overlaid 2-D decision region, scatter, and internals plots for the vowel-classification experiment.



**FIGURE 9.** Vowel-classification experiment of Figure 8: (a) profile and (b) histogram for the case in which the second input feature $x_2$ is set to 0.0 and the first input feature $x_1$ is swept from –2.0 to 4.0.

## Three LNKnet Applications

Lincoln Laboratory, the FBI, Rome Laboratory, the Air Force Institute of Technology (AFIT), and other research laboratories have used LNKnet software for many diverse applications. This section summarizes three such applications at Lincoln Laboratory. First, we describe a hybrid neural-network/hidden-Markov-model isolated-word recognizer that uses LNKnet RBF-classifier subroutines. Next, we describe experiments in which secondary testing with LNKnet MLP classifiers improved the wordspotting accuracy of a hidden-Markov-model wordspotter. Finally, we describe a software program that rapidly learns to replicate human game-playing strategy by using LNKnet MLP subroutines. In addition to these three examples, LNKnet software has facilitated the development of new pattern-classification algorithms, including the boundary hunting RBF classifier described in Reference 24.

### *Isolated-Word Recognition Using a Hybrid Neural-Network/Hidden-Markov-Model System*

Many researchers are using neural networks to estimate the local per-frame probabilities that are required in hidden-Markov-model (HMM) speech recognizers [25, 26]. Previously, these probabilities were estimated through the use of non-discriminant training with Gaussian and Gaussian-mixture probabilistic models. The understanding that network outputs are posterior probabilities allows the networks to be integrated tightly with HMM and other statistical approaches. Figure 11 shows a hybrid neural-network/ HMM speech recognizer that combines radial basis function (RBF) neural networks and HMMs for the speech recognition of isolated words [26, 27]. We have developed this system by integrating LNKnet RBF-classifier subroutines with HMM software. The RBF networks in the system produce posterior probabilities representing the probability that a specific subword acoustic speech unit occurred, given input features from a 10-msec input speech frame.

By dividing the network outputs by the class prior probabilities, the system normalizes the outputs to be scaled likelihoods. (Note: The prior probabilities are the estimated frequency of occurrence of each speech
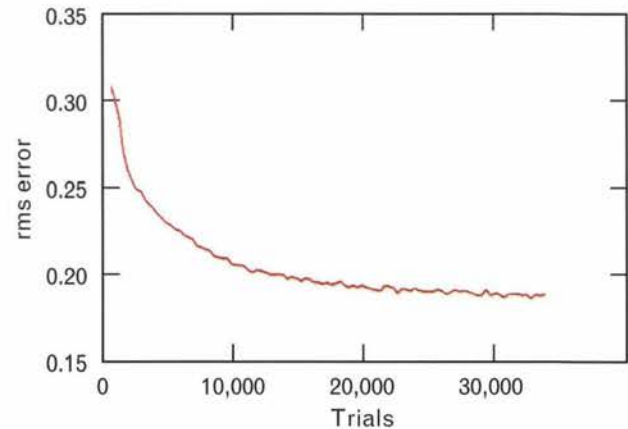


**FIGURE 10.** Plot of rms error during training for the vowel-classification experiment of Figure 8.

sound.) The scaled likelihoods can then be fed to Viterbi decoders [28] that perform nonlinear time alignment to compensate for varying talking rates and differences in word pronunciation. The Viterbi decoders align the input frames with the class labels of subword speech units and specify the correct labels for all frames. One Viterbi decoder for each keyword to be detected produces an accumulated output score for every keyword at the end of each input utterance.

We tested the hybrid recognizer on a difficult talker-
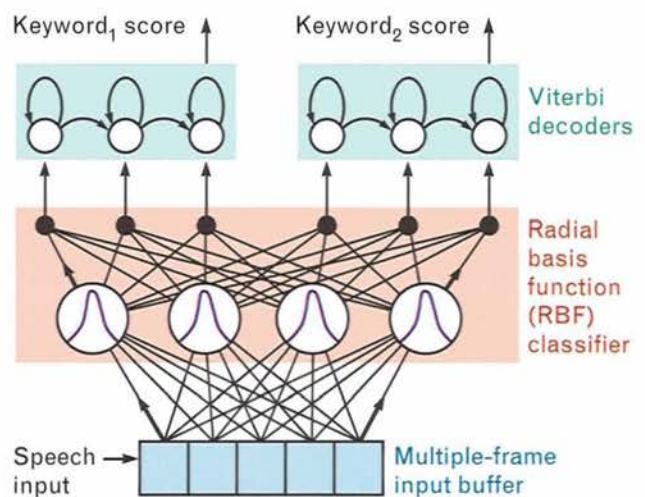


**FIGURE 11.** A hybrid isolated-word recognizer that uses radial basis function (RBF) networks to generate posterior probabilities for statistical Viterbi decoders [28]. In this example, there are three states (the beginning, middle, and end) for the keyword in each decoder.
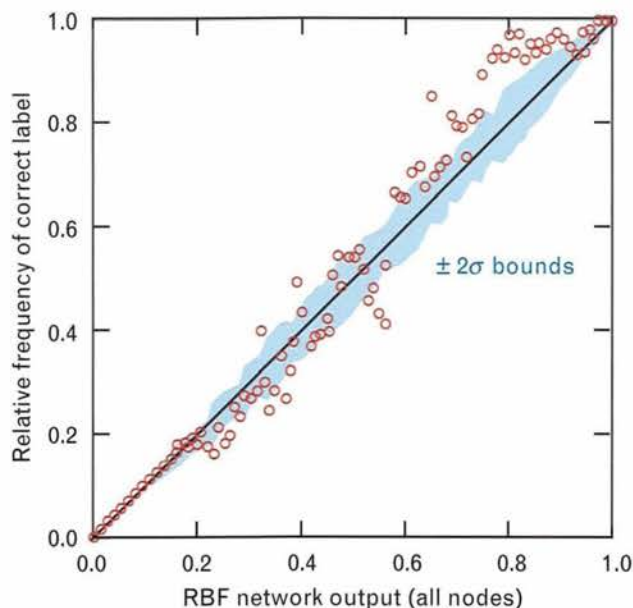
**FIGURE 12.** Comparison of RBF network outputs to posterior probabilities.

independent recognition task in which the goal was to distinguish between the nine spoken letters of the alphabet containing the long vowel "e" (i.e., the letters b, c, d, e, g, p, t, v, and z). For this task, the system achieved error rates that were lower than those obtained by a state-of-the-art high-performance Gaussian tied-mixture recognizer with an equal number of trainable parameters [26, 27].

The good performance achieved by this and other hybrid recognizers suggests that the network outputs do closely approximate posterior probabilities. We evaluated the accuracy of posterior-probability estimation by examining the relationship between the network output for a given input speech frame and the probability of classifying that frame correctly. If network outputs do represent posterior probabilities, then a specific network output value (between 0.0 and 1.0) should reflect the relative frequency of occurrence of correct classifications of frames that produced that output value. Furthermore, if posterior-probability estimation is exact, then the relative frequency of occurrence of correct classifications should match the network output value exactly.

Because there was only a finite quantity of data, we partitioned the network outputs into 100 equal-sized bins between 0.0 and 1.0. The values of RBF outputs were then used to select bins whose counts were incremented for each speech frame. In addition, the single correct-class bin count for the one bin that corresponded to the class of the input pattern was incremented for each frame. We then computed the ratio of the correct-class count to the total count and compared that ratio to the value of the bin center. For example, our data indicated that for the 61,466 frames of the speech utterances that were used for training, outputs of the RBF networks in the range from 0.095 to 0.105 occurred 29,698 times, of which 3067 instances were correct classifications. Thus the relative frequency of correct labeling for this particular bin was 0.103, which was close to 0.10, the bin center.

A plot of the relative frequencies of correct labeling for each bin versus the bin centers gives a measure of the accuracy of posterior-probability estimation by the RBF neural networks. Figure 12 shows the measured relative frequency of correct labeling for the RBF networks and the $2\sigma$ bounds for the binomial standard deviation of each relative frequency. Note that the relative frequencies tend to be clustered around the diagonal and many are within the $2\sigma$ bounds. This result suggests that network outputs are closely related to the desired posterior probabilities.

### Secondary Testing for Wordspotting

In secondary testing, a neural network is used to correct the more frequent confusions made by a simpler, more conventional classifier or expert system. Secondary testing can provide improved performance if (1) the confusions are limited to a small number of input classes, (2) there is sufficient training data for these classes, and (3) the input features provide information useful in discriminating between these classes. One application for secondary testing is in wordspotting.

Recent research at Lincoln Laboratory, Bell Laboratories, and other speech research sites [28–30] has begun to focus on the use of wordspotters to handle unconstrained verbal interactions between humans and machines. Wordspotters do not try to recognize every input, but instead they try to determine when certain keywords or phrases occur. Thus extraneous noise and words that do not change the meaning of the verbal input can be ignored and an open micro-

phone (i.e., a microphone that is left on continuously) can be used. Potential commercial applications of wordspotting include the sorting and selecting of voice mail by talker and topic, the voice control of consumer products, the use of voice-activated call buzzers for hospital patients to summon nurses, and the replacement of telephone operators for simple functions.

We have applied secondary testing to the output of a state-of-the-art talker-independent HMM word-spotter developed at Lincoln Laboratory [28, 31]. Our experiments used the Road Rally speech database containing telephone conversations between talkers performing a navigational task with road maps. To create a training dataset, we ran the HMM wordspotter on the Road Rally conversations and extracted speech segments that corresponded to putative hits for the following 20 keywords: Boonsboro, Chester, Conway, interstate, look, Middleton, minus, mountain, primary, retrace, road, secondary, Sheffield, Springfield, thicket, track, want, Waterloo, Westchester, and back-track. The putative hits represented speech frames where the 20 keywords might have occurred. Features derived from the average cepstra at the beginning,

middle, and end of each putative hit were then extracted to create training patterns for LNKnet. (Note: Cepstra are found by taking the fast Fourier transform [FFT] of the windowed input speech, followed by taking the smoothed log magnitude of the FFT, and then by taking the inverse FFT of the resulting quantity.) Next, we used LNKnet neural networks for the further classification of the putative hits as valid putative hits or false alarms, as shown in Figure 13. In this approach, one neural network classifier was trained to discriminate between correct hits and false alarms for each word that generated an excessive number of false alarms. Putative hits from words that generated few false alarms were passed on without processing.

We performed all experiments with the LNKnet point-and-click interface. For the classifier development with LNKnet, cross-validation testing was chosen because there were so few training patterns for most keywords. Using $N$-fold cross-validation testing, LNKnet split the training data into $N$ equal-sized folds and performed $N$ experiments, each time training with $N - 1$ folds and testing with the remaining fold. LNKnet performed both the splitting of the
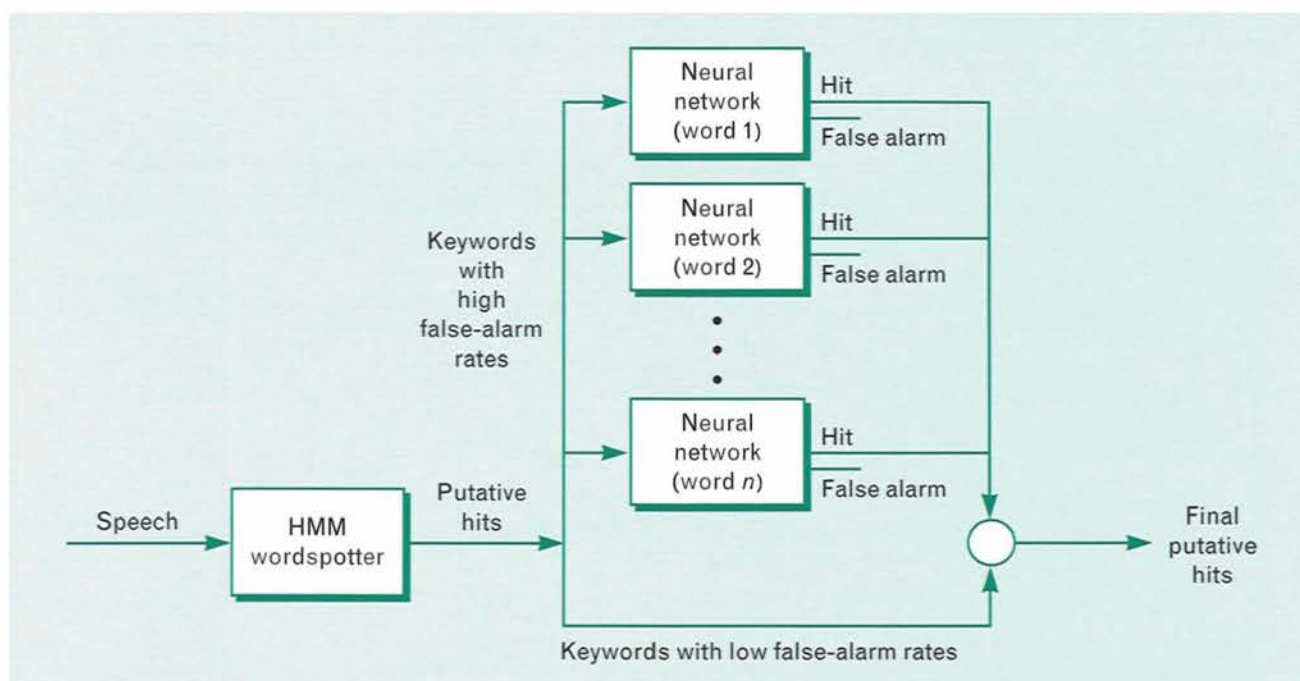


**FIGURE 13.** Secondary testing for wordspotting. The neural networks are used to distinguish between the valid putative hits and false alarms that the hidden-Markov-model (HMM) wordspotter has detected.
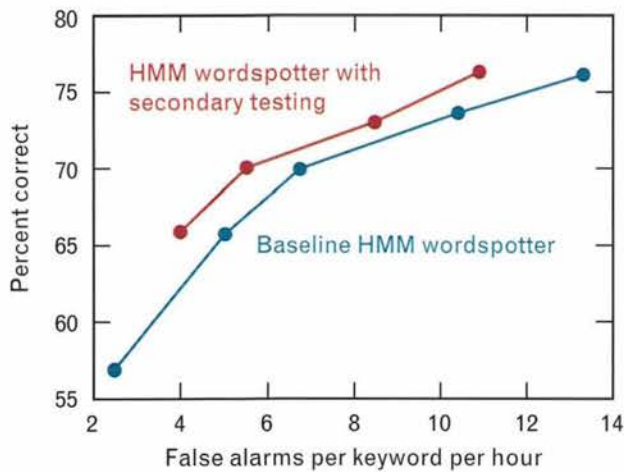
**FIGURE 14.** Wordspotting detection accuracy versus number of false alarms per keyword per hour generated with and without neural network secondary testing.

data and the cross-validation testing automatically. The average error rate that occurred during the testing of the *N* remainder folds was a good estimate of the generalization error on unseen data. The experiments suggested that multilayer perceptrons trained with back-propagation and with one hidden layer provided the best performance with the limited numbers of putative hits available for training. Furthermore, the average cepstra extracted from the beginning and end of each putative hit were found to provide good discrimination.

We performed further secondary-testing experiments with the same database and keywords as part of a Defense Advanced Research Projects Agency

(DARPA) workshop on speech evaluation held in Washington, D.C., on 10 and 11 March 1992. Reference 31 contains details of this evaluation and Figure 14 summarizes the results. The blue curve in the figure shows the detection accuracy of the primary HMM wordspotter as a function of the number of false alarms per keyword per hour. Note that the detection accuracy increases as we allow the number of false alarms to increase. The red curve in the figure shows the increase in detection accuracy achieved with neural networks used for secondary testing. One network for each of the four words that produced many false alarms was used to reclassify putative hits produced by the primary wordspotter. Overall, this postprocessing reduced the false-alarm rate by an average of 16.4%, thus demonstrating that neural networks can be used effectively as wordspotter postprocessors. Further analyses showed that the extra computational overhead required by secondary testing was much less than 5%.

### Learning a Game-Playing Strategy from a Human Player

Neural network classifiers can learn to reproduce the responses of human experts to new situations in tasks as diverse as driving a van [32] and playing backgammon [33]. An example of this type of learning is *netris,* a program that we created using LNKnet MLP-classifier subroutines. Netris learns the strategy that a human uses to play a modified version of *Tetris,* a popular computer game.
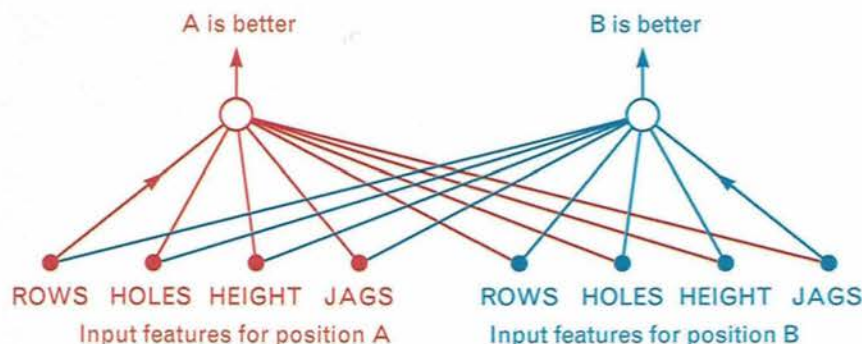


**FIGURE 15.** Neural network used to learn a human player's preferences for positioning pieces in the computer game *Tetris.*

In *Tetris,* different-shaped pieces appear one by one at the top of a rectangular playing grid and fall towards the bottom of the grid. A player must rotate (in 90° increments) and move (either left or right) each piece such that the pieces form complete solid rows across the bottom of the grid. The solid rows disappear, making room for more pieces, and points are awarded for each solid row. If the player is unable to complete solid rows across the bottom of the grid, the playing field will begin to fill up. The game ends when gridlock occurs at the top of the playing field and no new pieces have any room to fall. (Note: Readers who are unfamiliar with *Tetris* may look ahead to Figure 16, which contains two examples of playing fields.)

The netris program allows a human to play *Tetris* while simultaneously training a neural network to play in an adjacent screen. The network is trained with LNKnet subroutines to try to mimic the human player's decisions. During the training process, the move selected by the human for each falling piece is paired with all other permissible moves, thus creating multiple training patterns. A *preference network* trained with these patterns can then be used to select moves for new pieces in a different playing grid. The preference network finds the best move by comparing pairs of all permissible moves, always retaining the move that is judged better. This process requires only $N$ comparisons (given $N$ possible moves) because the rejected move is dropped after each comparison and only the winning move is kept for comparison with the remaining moves. The network trains rapidly (enabling real-time learning) and reproduces a human player's decisions accurately. If the human makes consistently good moves, the network will gradually learn to play better and better.

Initial experiments led to the simple position-preference network shown in Figure 15. The network has eight linear input nodes, two sigmoid output nodes, and 18 weights (including two bias weights not shown). For the input features to the network, a human player has selected certain important characteristics of the piece distribution at the bottom of the *Tetris* playing field. The input features selected are the number of rows completed by the falling piece (ROWS), the number of holes created below the piece



| 0 rows completed | 18 rows completed |
| 6 pieces dropped | 50 pieces dropped |
| (a) | (b) |
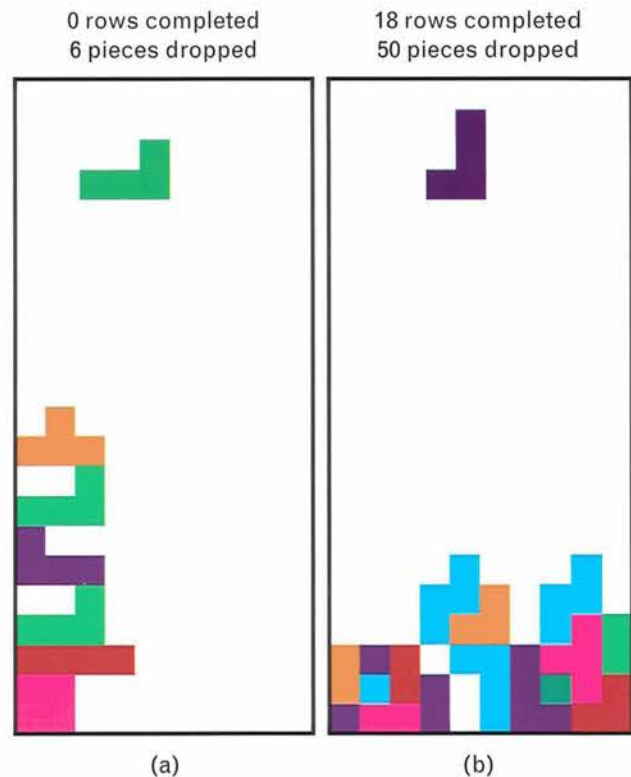
**FIGURE 16.** Configuration of pieces by preference network with (a) no training and (b) after training on 50 pieces that were positioned by a human player in the popular computer game *Tetris.*

(HOLES), the maximum height of the piece (HEIGHT), and the variability in the contour formed by the tops of all pieces (JAGS). These four input features are provided for the two permissible and unique moves (A and B) that are being compared, and the network determines whether A or B is preferred by selecting the move corresponding to the output node with the highest value.

Figure 16(a) shows an example of how pieces pile on top of one another without forming rows when the preference network has not been trained. Without such training, gridlock occurs in the playing field after about 9 to 13 pieces have fallen. Figure 16(b) shows how the pieces fall more purposefully after the network has been trained with only 50 decisions made by an unskilled human player. With such training, 18 rows have been completed after 50 pieces have fallen, and the strategy used by the human player is being imitated by the preference network.

## Summary

A software package named LNKnet simplifies the task of applying neural network, statistical, and machine-learning pattern-classification algorithms in new application areas. LNKnet classifiers can be trained and tested on separate data or tested with automatic cross-validation. The point-and-click interface of the software package enables non-programmers to perform complex pattern-classification experiments, and structured subroutine libraries allow classifiers to be embedded in user application programs. LNKnet has been used successfully in many research projects, including the development of a hybrid neural-network/hidden-Markov-model isolated-word recognizer, the improvement of wordspotting performance with secondary testing, and the learning of a human's game-playing strategies. LNKnet software has also been applied in other diverse areas, including talker identification, talker-gender classification, hand-printed-character recognition, underwater and environmental sound classification, image spotting, seismic-signal classification, medical diagnosis, galaxy classification, and fault detection.

LNKnet is currently available through the MIT Technology Licensing Office.

## Acknowledgments

# REFERENCES

1. M.D. Richard and R.P. Lippmann, "Neural Network Classifiers Estimate Bayesian *a Posteriori* Probabilities," *Neural Computation* **3**, 461, 1992.
2. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis* (John Wiley, New York, 1973).
3. K. Fukunaga, *Introduction to Statistical Pattern Recognition* (Academic Press, New York, 1972).
4. J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation* (Addison-Wesley, Reading, MA, 1991).
5. D.R. Hush and B.G. Horne, "Progress in Supervised Neural Networks," *IEEE Signal Process. Mag.* **10**, 8 (Jan. 1993).
6. R.P. Lippmann, "An Introduction to Computing with Neural Nets," in *Neural Networks: Theoretical Foundations and Analysis*, ed. C. Lau (IEEE Press, New York, 1992).
7. R.P. Lippmann, "A Critical Overview of Neural Network Pattern Classifiers," in *Neural Networks for Signal Processing, Proc. of the 1991 IEEE Workshop, Piscataway, NJ, 1991*, eds. B.H. Juang, S.Y. Kung, and C.A. Kamm, p. 266.
8. B.G. Batchelor, ed., *Pattern Recognition: Ideas in Practice* (Plenum Press, New York, 1978).
9. J.B. Hampshire II and B.V.K. Vijaya Kumar, "Why Error Measures are Sub-Optimal for Training Neural Network Pattern Classifiers," in *IEEE Proc. 1992 Int. Joint Conf. on Neural Networks, Baltimore, 7–11 June 1992*, p. IV-220.
10. J.B. Hampshire II and A.H. Waibel, "A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks," in *IEEE Trans. Neural Netw.* **1**, 216 (1990).
11. N.J. Nilsson, *Learning Machines* (McGraw-Hill, New York, 1965).
12. J.A. Hartigan, *Clustering Algorithms* (John Wiley, New York, 1975).
13. W.Y. Huang and R.P. Lippmann, "Comparisons between Neural Net and Conventional Classifiers," in *Proc. 1st Int. Conf. on Neural Networks, San Diego, 21–24 June 1987*, p. IV-485.
14. L.N. Kukolich and R.P. Lippmann, "LNKnet User's Guide," MIT Lincoln Laboratory Technical Report in press, 1993.
15. Y.C. Lee, *Classifiers: Adaptive Modules in Pattern Recognition Systems*, S.M. Thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA (1989).
16. Y.C. Lee and R.P. Lippmann, "Practical Characteristics of Neural Network and Conventional Pattern Classifiers on Artificial and Speech Problems," in *Advances in Neural Information Processing Systems 2*, ed. D.S. Touretzky (Morgan Kaufmann, San Mateo, CA, 1990).
17. R.P. Lippmann, "Pattern Classification Using Neural Networks," in *IEEE Commun. Mag.* **27**, 47 (Nov. 1989).
18. K. Ng and R.P. Lippmann, "A Comparative Study of the Practical Characteristics of Neural Network and Conventional Pattern Classifiers," *Technical Report 894*, MIT Lincoln Laboratory (19 Mar. 1991).

19. K. Ng and R.P. Lippmann, "A Comparative Study of the Practical Characteristics of Neural Network and Conventional Pattern Classifiers," in *Neural Information Processing Systems* 3, eds. R. Lippmann, J. Moody, and D. Touretzky (Morgan Kaufmann, San Mateo, CA, 1991), pp. 970–976.

20. I. Guyon, I. Poujand, L. Personnaz, G. Dreyfus, J. Denker, and Y. Le Cun, "Comparing Different Neural Network Architectures for Classifying Handwritten Digits," *Proc. Int. Joint Conf. on Neural Networks, Washington, DC, 1989*, p. II.127.

21. S.M. Weiss and C.A. Kulikowski, *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems* (Morgan Kaufmann, San Mateo, CA, 1991).

22. T.W. Parsons, *Voice and Speech Processing* (McGraw-Hill, New York, 1986).

23. G.E. Peterson and H.L. Barney, "Control Methods Used in a Study of Vowels," in *J. Acoust. Soc. Am.* 24, 175 (1952).

24. E.I. Chang and R.P. Lippmann, "A Boundary Hunting Radial Basis Function Classifier Which Allocates Centers Constructively," in *Neural Information Processing Systems* 5, eds. S. Hanson, J. Cowan, and C. Giles (Morgan Kaufmann, San Mateo, CA, 1993), pp. 139–146.

25. N. Morgan and H. Bourlard, "Continuous Speech Recognition Using Multilayer Perceptrons with Hidden Markov Models," in *Proc. Int. Conf. on Acoustics, Speech and Signal Processing, Albuquerque, NM, 3–6 Apr. 1990*, p. I-413.

26. E. Singer and R.P. Lippmann, "Improved Hidden Markov Model Speech Recognition Using Radial Basis Function Networks," in *Neural Information Processing Systems* 4, eds. J. Moody, S. Hanson, and R. Lippmann (Morgan Kaufmann, San Mateo, CA, 1992), pp. 159–166.

27. E. Singer and R.P. Lippmann, "A Speech Recognizer Using Radial Basis Function Neural Networks in an HMM Framework," in *Proc. Int. Conf. on Acoustics, Speech and Signal Processing, San Francisco, 23–26 Mar. 1992*, p. I-629.

28. R.C. Rose, "Techniques for Information Retrieval from Speech Messages," *Linc. Lab. J.* 4, 45 (1991).

29. J.R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous Hidden Markov Model for Speaker Independent Word Spotting," in *Proc. Int. Conf. on Acoustics, Speech and Signal Processing, Glasgow, 23–26 May 1989*, p. I-627.

30. J.G. Wilpon, L.R. Rabiner, C.-H. Lee, and E.R. Goldman, "Automatic Recognition of Keywords in Unconstrained Speech Using Hidden Markov Models," in *IEEE Trans. Acoust. Speech Signal Process.* 38, 1870 (1990).

31. R.P. Lippmann and E. Singer, "Hybrid Neural-Network/HMM Approaches to Wordspotting," in *Proc. Int. Conf. on Acoustics, Speech and Signal Processing, Minneapolis, MN, 27–30 Apr. 1993*, p. I-565.

32. D. Pomerleau, "Rapidly Adapting Artificial Neural Networks for Autonomous Navigation," in *Neural Information Processing Systems* 3, eds. R. Lippmann, J. Moody, and D. Touretzky (Morgan Kaufmann, San Mateo, CA, 1991), pp. 429–435.

33. G. Tesauro, "Practical Issues in Temporal Difference Learning," in *Neural Information Processing Systems* 4, eds. J. Moody, S. Hanson, and R. Lippmann (Morgan Kaufmann, San Mateo, CA, 1992).
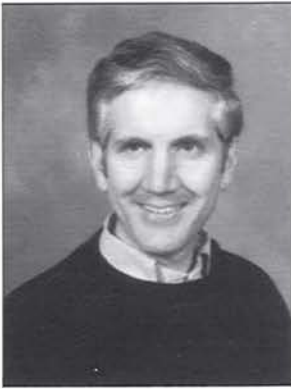
**RICHARD P. LIPPMANN**
was born in Mineola, New York, in 1948. He received a B.S. degree in electrical engineering from the Polytechnic Institute of Brooklyn in 1970, and an S.M. and a Ph.D. degree in electrical engineering from MIT in 1973 and 1978, respectively. His S.M. thesis dealt with the psychoacoustics of intensity perception, and his Ph.D. thesis with signal processing for the hearing impaired.

From 1978 to 1981, he was the Director of the Communication Engineering Laboratory at the Boys Town Institute for Communication Disorders in Children in Omaha, Nebraska. He worked on speech recognition, speech training aids for deaf children, sound alerting aids for the deaf, and signal processing for hearing aids. In 1981, he joined Lincoln Laboratory and is currently a senior staff member in the Speech Systems Technology Group, where his focus of research has been in speech recognition, speech I/O systems, and routing and system control of circuit-switched networks. His current interests include speech recognition, neural network algorithms, statistics, and human physiology, memory, and learning. Rich is a founding member of the Neural Information Processing Systems (NIPS) Foundation, which has sponsored the annual NIPS conference in Denver, Colorado, since 1988.

**LINDA C. KUKOLICH**
is an assistant staff member in the Speech Systems Technology Group, where her focus of research has been in algorithm and user-interface development for pattern classification, especially in the field of speech recognition. She received a B.S. degree in applied mathematics from MIT.

**ELLIOT SINGER**
received a B.S. degree in electrical engineering from Polytechnic University in 1971 and an S.M. degree in electrical engineering from MIT in 1974. He is a staff member in the Speech Systems Technology Group, where his focus of research has been in speech recognition and wordspotting. From 1975 to 1977, he was an instructor for the Department of Electrical Engineering and Computer Science at MIT. He is a member of Tau Beta Pi and Eta Kappa Nu.